Europäisches Patentamt	European Patent Office	Office européen des brevets

Bescheinigung Certificate Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

00124796.4

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

CERTIFIED COPY OF
PRIORITY DOCUMENT

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN
THE HAGUE, 15/05/01
LA HAYE, LE

NOV 2 1 2006

THIS PAGE BLANK (USPTO)

Europäisches Patentamt | European Patent Office | Office européen des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.:
Application no.: 00124796.4
Demande n°:

Anmeldetag:
Date of filing: 14/11/00
Date de dépôt:

Anmelder:
Applicant(s):
Demandeur(s):
International Business Machines Corporation
Armonk, NY 10504
UNITED STATES OF AMERICA

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:
Method and system for advanced restart of application servers processing time-critical requests

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: State: Pays:	Tag: Date: Date:	Aktenzeichen: File no. Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:
/

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE/TR
Etats contractants désignés lors du depôt:

Bemerkungen:
Remarks:
Remarques:

NOV 2 1 2006.

THIS PAGE BLANK (USPTO)

D E S C R I P T I O N

METHOD AND SYSTEM FOR ADVANCED RESTART OF APPLICATION SERVERS PROCESSING TIME-CRITICAL REQUESTS

EPO - Munich
17
14. Nov. 2000

BACKGROUND OF THE INVENTION

The invention relates to a data processing environment comprising one or more application servers processing requests, particularly time-critical requests, on behalf of at least one application and, more specifically, to a method and a system for controlling restart processing after termination of one or more resource managers that are involved in managing resources needed by an application server to fulfill application requests.

A typical data processing environment comprising an application server for processing time-critical transactions is shown in Fig. 1. An application server implements a collection of related services which are requested from applications via application clients. Applications, application servers, and application clients may be implemented by computer programs of any nature not limited to a specific type of implementation. Application, application server, and application client are only logical structures; for example an application server could be the application that requests services from another application server or even requests from itself.

Application servers typically run on the second tier of a three-tier system structure. In such a three-tier system structure the application client is on the first tier, requesting services from an application server on the second tier, which in turn requests services from backend applications that are located on the third tier. The three-tier system structure is a conceptual structure. That means, even if it is typical to deploy the different tiers onto different computers,

NOV 2 1 2006

there is no need to do so; the deployment of the complete structure onto a single computer is a completely valid approach. It should be noted that a three-tier structure is just a special case of an n-tier structure. For example, if users are interacting via a Web browser, then the Web browser would be considered running on tier 0 and the associated web server which runs the application requesting services from the application server is considered running on tier 1.

Application servers are typically stateless, that means they store any necessary state information into a persistent data storage. State information is data that the application server maintains between two subsequent requests from a client so that the individual requests from the application requesting services can be correlated.

Application servers in general run the processing of requests as transactions in the sense of ACID transactions; that means all requests are either processed completely or not at all. A thorough representation of transaction technology is given by Jim Gray and Andreas Reuter, Transaction Processing, Morgan Kaufmann Publishers, San Mateo, 1993. In the case of an error, the transaction is aborted, all resources are reset to the state they have been before the transaction was started. If the access to the relational database that maintains the application server state, for example, fails, all changes made to any of the involved resources are backed out; for example, the message in a message queue that represents the request is put back into the queue from where it was read. This requires that all resources that are used by the application server for persistent data are managed by resource managers that can participate in transactions.

If any one of the involved software components fails, all components must be brought back to the state when the failure of the one or more components occurred so that processing can

continue. The process of bringing a failed component back is called a restart process. The time it takes to restart a component depends how much work needs to be undone and redone to come up to reestablish the state of the component at the time of failure.

Resource managers maintain a log that contains entries about all changes that were applied to their managed resource. The log is typically maintained on highly available disks, that for example exploit RAID technology. In addition, the resource managers periodically take snapshots of their current state and writes this information as a checkpoint into the log. This information is used for recovery from resource manager failures, such as the abnormal termination of the resource manager or the loss of the media on which the resource manager maintains the resource. Recovering from an abnormal termination of the resource manager is called crash recovery, recovering from a media failure is called media recovery. It should be noted, that media recovery is no longer considered an issue in resource manager recovery due to the high availability of the disks on which the data is maintained and is therefor not considered in the present invention. The component of resource managers that is responsible for recovering from failures is typically called the restart manager. When the resource manager is restarted after the crash, the restart manager uses the checkpoint information and the individual log entries to reconstruct the latest state; that means the restart component needs to figure out which transactions need to be aborted, such as those executing when the crash occurred, and which committed updates need to be redone. This is done by locating the checkpoint record and processing all log records that are coming after the checkpoint record. Thus, the frequency with which checkpoint records are taken, determines the amount of processing that is required to restart the failing component after a failure.

In the prior art approaches different resource managers have

different policies when to write a checkpoint record. These policies are typically specified via some global settings, such as the number of processed messages in a message queueing system or the time period between two subsequent checkpoints in a relational database management system. These settings may be fixed, that means they can not be changed by the user, as for example in the message queueing system MQSeries of the present applicant, where the number of processed messages is set to 1000; or variable, as for example in the relational database management system DB2 of the present applicant, where the time between checkpoints can be set by the user.

A resource manager typically serves multiple applications; for example multiple different application servers. In this case, the log is used for all changes caused by all applications. Some resource managers however such as DB2 allow to run multiple instances. An instance consists of all the information needed to support users; it has its own catalog for managing metadata, own log, and appropriate user data. A particular application server can be associated with a particular instance. In this case, the log is only used for the operations of that application server.

The checkpoint record frequency settings for each of the involved resource managers determines how long it will take to recover from the crash of one or more of the resource managers. As each participating resource manager takes checkpoints independently, the maximum restart time of the application server can be calculated as the restart time of the resource manager with the longest restart time plus the restart time of the application server itself.

Typically, the restart time is obtained by running simulations with the application server. Since simulation never matches real-life situations, the estimated restart time is only an approximate value.

Taking checkpoints is not only a time-consuming operation, it also slows down processing of requests as the resource manager must dedicate processing and I/O resources for taking the checkpoint. Since there is no correlation between the processing of the application server and the frequency of checkpoint taking by the involved resource managers, checkpoints may be taken, even if not required.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide an above described method and system which guarantee that in the case of a failure of the environment, such as a crash of one or more of the resource managers, the request processing time of the application server does not exceed a pre-specified, in particular a user-defined, request processing time.

It is another object to provide such a method and system which enable an application server that processes time-critical requests to guarantee that a user-specified request processing time is not exceeded, even if any of the resource managers that manage resources needed to satisfy the requests terminates abnormally.

It is yet another object to provide such a method and system which minimize the number of checkpoints being taken by the resource managers involved in running the application server transactions.

These objects are solved by the features of the independent claims. Advantageous embodiments are subject matter of the subclaims.

The above specified problems are solved by the method according to the invention by having the resource managers taking

checkpoints in a frequency that guarantees that the restart of the application server including the restart of the resource managers including the processing time of the request does not exceed the user specified request processing time. Even in the case of failure of any of the involved resource managers, it is achieved that the time it takes to have the requests processed again is shorter than a specified request processing time. Thus the proposed mechanism ensures that the resource managers take checkpoints in such a frequency that the time it takes to have the application server processing requests again is shorter than the specified request processing time.

The concept underlying the invention is based on the observation that the frequency with which checkpoint records are taken determines the amount of processing that is required to restart a failing component after a failure.

The method according to the invention preferably specifies two mechanisms of controlling when a checkpoint needs to be written, one being a load-controlled mechanism and the other being a restart-time-controlled mechanism.

In both cases, specification of the request processing time may either be hard coded in the application server or provided to the application server as a parameter. As an additional option, the application server may support changing of the request processing time during operation, for example, by offering an appropriate application programming interface, or some graphical/textual interface.

In the load-controlled method, the application server knows the amount of data that each resource manager writes into its log and the amount of time it takes the resource manager to process the log in the case of a restart. Thus the application server knows at any time the currently necessary restart time of each of the resource manager and, based on that information, requests

the taking of checkpoints from the resource managers so that the specified request processing time could even be met in the case of a failure of one or more resource managers. The load-controlled method is based on the assumption that the resource managers provide the capability for calling applications to request the taking of checkpoints.

In the restart-time-controlled method, the resource manager provides the capability for the specification of a restart time. This allows the application server to determine from the user specified request processing time the appropriate restart time for each of the involved resource managers. This information is handed over to the resource manager when the application server connects to the resource manager the first time or when the user specified request processing time is changed.

Thereupon, a further advantage of the proposed mechanism is that checkpoints have only to be taken if necessary, resulting in an improved throughput of the application server.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will now be described in the following in more detail by way of embodiments whereby reference is made to the accompanying drawings. In the drawings,

Fig. 1 is a schematic block diagram depicting an application server according to the prior art where the present invention can be applied;

Fig. 2 is a schematic block diagram depicting the internal structure and transaction boundaries of a message-based application server according to the prior art where the present invention can be applied;

NOV 2 1 2006

Fig. 3 is a flow diagram illustrating the user controlled resource manager checkpoint controlling mechanism according to the invention;

Fig. 4 is a flow diagram illustrating detailed method steps of the load-controlled checkpoint mechanism proposed by the invention; and

Fig. 5 shows exemplary actions performed by an application server where a resource manager supports a user-defined restart time.

DETAILED DESCRIPTION OF THE DRAWINGS

Fig. 1 shows the typical structure of an application server. A number of applications request services from the application server via application clients. The application server 100 implements a collection of services which are requested from applications 110, 112, and 114 via application clients 120, 122, and 124. The applications 110, 112, and 114, the application server 100, and the application clients 120, 122, and 124 may be implemented by computer programs of any nature not limited to any specific type or implementation. The applications 110, 112, and 114, the application server 100, and the application clients 120, 122, and 124 are only logical structures. For example, the application server 100 can be the application 110 itself that requests services from another application server (not shown here) or even requests services from itself. As the application server 100 is stateless, all information that is needed between subsequent requests from the applications 110, 112, and 114 are kept in a datastore 140.

Fig. 2 comprises a particular implementation of an application server, namely a message-based application server 200, that means the communication between the application client 210 and the application server 200 is based on asynchronous reliable

message exchange. But it is understood hereby that the invention is not limited to a certain communication paradigm. Fig. 2 is just used to illustrate how an application server processes requests as transactions; the transaction boundaries are indicated by dashed lines.

When an application 220 requests a service from the application server 200, the application client 210 puts a message reflecting the application's request into the application server input queue 280. This is done as a transaction 260, labeled Transaction 1, so that when the request has been successfully put into the application server's input queue, the application's request will be honored.

The application server 200 processes requests within a transaction 240, labeled Transaction 2. The transaction consists of reading the request message from the input queue 280, performing some processing of that message, saving processing state information persistent into a data store 230, and putting the response message into the application server's output queue 285.

The application 220 gets this response by the application client 210 reading the response message from the application server's output queue. This action is also run as a transaction 250, labeled transaction 3.

In the application server's processing of a request message and generating a response message, multiple resources are accessed and possibly modified. Since multiple resources and thus multiple resource managers are involved in such a transaction, the transaction must be coordinated via a transaction manager 270 typically using a two-phase-commit (2PC) protocol. The individual resources are typically managed by different resource managers. For example, the application server's input and output queue 280, 285 are managed by a (not shown) message queueing

NOV 2 1 2006

system and the data store 230 by a (not shown) relational database management system. The involvement of multiple resource managers mandates a transaction manager which coordinates the processing of the individual resource managers when the transaction needs to be committed or aborted. The 2PC protocol is the protocol that transaction managers typically exploit to coordinate commmit/abort processing with the resource managers involved in a particular transaction. Further information can be found in the book "Transaction Processing: Concepts and Techniques", authored by Jim Gray and Andreas Reuter and published by Morgan Kaufmann Publishers, Inc., 1993 which is regarded to be entirely incorporated herein by reference.

Referring again to Fig. 2, the application server 200 processes each individual client request as a transaction. The term transaction should not be understood as limited to the classical definition of transactions, that means having ACID properties, but to all units of work concepts that provide some form of transaction control by weakening one or more of the ACID properties.

It is further noted that, although the above described description is related to an application server, the scope of the present invention is by no means limited to such and application server and moreover can be used for any other application which performs the described interaction patterns with resource managers.

Fig. 3 shows the processing of a request by the resource manager if the resource manager supports user-specified checkpoint processing based on the present invention. It assumes that one or a multitude of users has specified a maximum restart time, or restart time for short, for the resource manager and the resource manager has stored this information internally. It is noted that the user or multitude of users, alternatively, can specify two or more restart times wherein the proposed mechanism

takes the minimum restart time for generating checkpoints. When the resource manager processes the next request 300, it first determines the current settings for the user specified restart time 310. This restart time must not be exceeded in case of failure. Next, the resource manager determines the restart time needed for the current request 320 and the potential previous requests, which would participate in restart processing since the last checkpoint processing, and then calculates the new restart time 330. If the new restart time would exceed the specified restart time effect of the request on the overall restart time 350, a checkpoint is taken 340. Then the request is processed 360 and the new restart time is calculated 370. It should be noted, that Fig. 3 is for illustration purpose only; actual implementations are most likely more sophisticated.

In the following, two different embodiments of the invention are described in more detail.

Load-controlled checkpointing

It is hereby assumed that resource managers provide the capability for applications to request checkpoints. How the resource managers externalize this capability is immaterial, whether it is the sending of a message to queue or the invocation of the request via an application programming interface.

Fig. 4 shows the processing of a request by the application server using load-controlled checkpointing according to the present invention. It is assumed that the application server has knowledge about the amount of log data written by each resource manager and the amount of time it takes to process the log during recovery. It further assumes that the user has specified the maximum request response time (not the restart time!), or response time for short, for the application server and the application server has stored this information internally.

NOV 2 1 2006

When the application server processes the next request 400, it first determines the current settings for the user specified request response time 410. This request response time must not be exceeded in case of failure. Next, the application server performs a set of actions (indicated by loop 470) for all resource managers. First, the application server determines the restart time needed for the current request 420 and the potential previous requests, which would participate in restart processing since the last checkpoint processing, and then calculates the new restart time 430. If the new restart time would exceed the restart time necessary to keep within the requested response time 450, the resource manager is requested to take a checkpoint 440. The mapping the request response time to the restart time takes into effect additional processing that is associated with the restart after a failure, such as starting the application server or reattaching to the failing resource manager. After all resource managers have been processed, the request is processed 460 and the new restart time is calculated for all resource managers 470. It should be noted, that Fig. 4 is for illustration purpose only; actual implementations are most likely more sophisticated.

If the resource manager does not support the notion of a resource manager instance, then the resource manager must provide the capability to assign a separate log to a particular application. In addition, the resource manager must provide the capability that applications can request that checkpoints are taken only for a particular log. When a crash occurs, then the resource manager must process this log before any other log, unless there is no time penalty when processing multiple logs in parallel.

Expressed in short, in the load-controlled checkpointing approach the above teaching is embodied within an application server which controls the checkpointing frequency based upon guaranteed response time requirements on behalf of the

underlying resource managers. This approach permits to reflect, in addition, the processing required to restart the application server itself.

Restart time controlled checkpointing

The previous approach, the load-controlled checkpoint approach, requires that the application server has a deep understanding of the logging and restart operations of each of the involved resource managers. In particular, the metrics associated with logging and restart need to be changed whenever the resource manager is changed.

In order to increase performance and throughput of the application server(s), in the present embodiment, the resource manager(s) themselves keep track of the restart time instead of the application server. The shown resource manager externalizes the capability for applications to set the restart time and the resource manager then takes a checkpoint whenever the specified restart time is reached. Fig. 3 shows how a resource manager could implement this capability.

Fig. 5 shows exemplarily the actions that the application server needs to take if the resource manager supports a user-defined restart time. In a first step, the application server obtains the user specified maximum request response time 500, or request response time for short. The application server then performs a set of actions (indicated by a loop) for all resource managers. The first step 510 in the set of actions is to calculate from the specified request response time the appropriate restart time for the resource manager. Transformation of the request response time to the restart time of the resource manager is necessary to cope with additional processing needs, such as the start up of the application server itself. In the second step 520, the application server hands over this restart time to the resource manager. The resource manager itself would then execute the code

shown in Fig. 3 if called by the application server. It should be noted, that Fig. 5 is for illustration purpose only; actual implementations are most likely more sophisticated.

EPO - Munich
17
14. Nov. 2000

C L A I M S

1. A method of dynamically controlling restart processing for recovery of at least one resource manager administrating at least one resource, said resources being accessible by at least one application to fulfill application requests,

said method being characterized by dynamically controlling the restart processing based upon a current maximum restart time by

a first step of dynamically calculating a current restart time required to restart the resource manager in case of a potential termination of said resource manager or potential recovery of said resource; and

a second step of dynamically initiating the resource manager to take a checkpoint whenever the current restart time exceeds said current maximum restart time.

2. Method according to claim 1, wherein, in case a new request to modify a resource arrived, the first step of dynamically calculating comprises to calculate said current restart time reflecting the new request and all accumulated requests since a last checkpoint, which would participate in the restart processing.

3. Method according to claim 2,

wherein the method is executed by the resource manager; and

wherein the current maximum restart time is specified to the resource manager by the application.

4. Method according to claim 3,

NOV 2 1 2006.

wherein, if a multitude of applications each specified an individual maximum restart time, said resource manager selects a smallest of said individual maximum restart times as said current maximum restart time.

5. Method according to claim 4,

wherein said application is an application server and/or an application client.

6. Method according to claim 2, wherein the method is executed by the application representing an application server said application server also administrating a request-load-log of said all accumulated requests since said last checkpoint for dynamically calculating said restart time.

7. Method according to claim 6, wherein the method is executed by the application server for a multitude of resource managers attached to the application server.

8. Method according to any of the claims 3 to 7,

wherein at least one application client specifies to said application server a maximum request response time defining a time not to be exceeded for application requests even in case of termination or recovery of said resource manager; and

wherein said application server is converting set maximum request response time to said current maximum restart time by subtracting from said maximum request response time processing time of further activities associated with said restart processing.

9. Method according to claim 8, wherein said further activities

comprise a processing time required to restart said application

server; and/or

comprise a processing time required for re-attaching said application server to said resource manager.

10. A resource manager comprising means adapted for carrying out the steps of the method according to anyone of the preceding claims 1 to 9.

11. Resource manager according to claim 10, wherein said resource manager offers checkpoint initiation means to initiate said resource manager to take a checkpoint.

12. An application server comprising means adapted for carrying out the steps of the method according to anyone of the preceding claims 1, 2, 6-9.

13. Application manager according to claim 12, wherein said application server accesses checkpoint initiation means offered by the resource manager to initiate to take a checkpoint.

14. Application server according to claim 12, comprising means for changing said maximum restart time or said maximum request response time dynamically during operation for instance by a corresponding application programming interface (API), by a graphical user interface (GUI), by a textual user interface (TUI) or the like.

15. A data processing program for execution in a data processing system comprising software code portions for performing a method according to anyone of the preceding claims 1 to 9 when said program is run on said computer.

16. A computer program product stored on a computer usable medium, comprising computer readable program means for causing a computer to perform a method according to anyone of the

NOV 2 1 2006

preceding claims 1 to 9 when said program is run on said computer.

A B S T R A C T

An application server that processes time-critical requests must guarantee that a user specified request processing time is not exceeded. Said specified request processing time must not be exceeded even if any of the resource managers, that manage resources needed to satisfy the requests, terminates abnormally.

The present invention proposes a mechanism that ensures that the resource managers take checkpoints in such a frequency, that even in the case of failure of any of the involved resource managers, the time it takes to have the requests processed again is shorter than a specified request processing time.

An additional advantage of the present invention is that checkpoints are only taken if necessary, resulting in improved throughput of the application server. (Fig. 3)

EPO - Munich
17
14. Nov. 2000

NOV 2 1 2006

THIS PAGE BLANK (USPTO)

(Drawings)

EPO - Munich
17
14. Nov. 2000

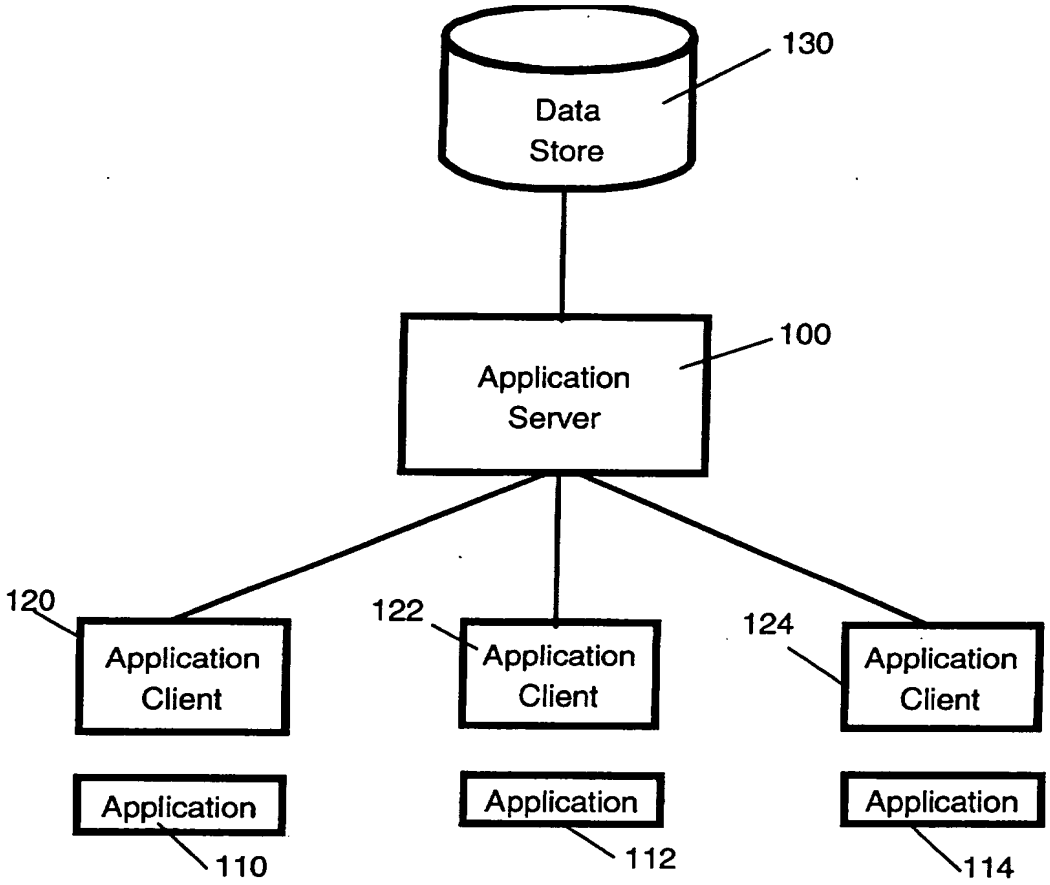

FIG. 1

NOV 2 1 2006

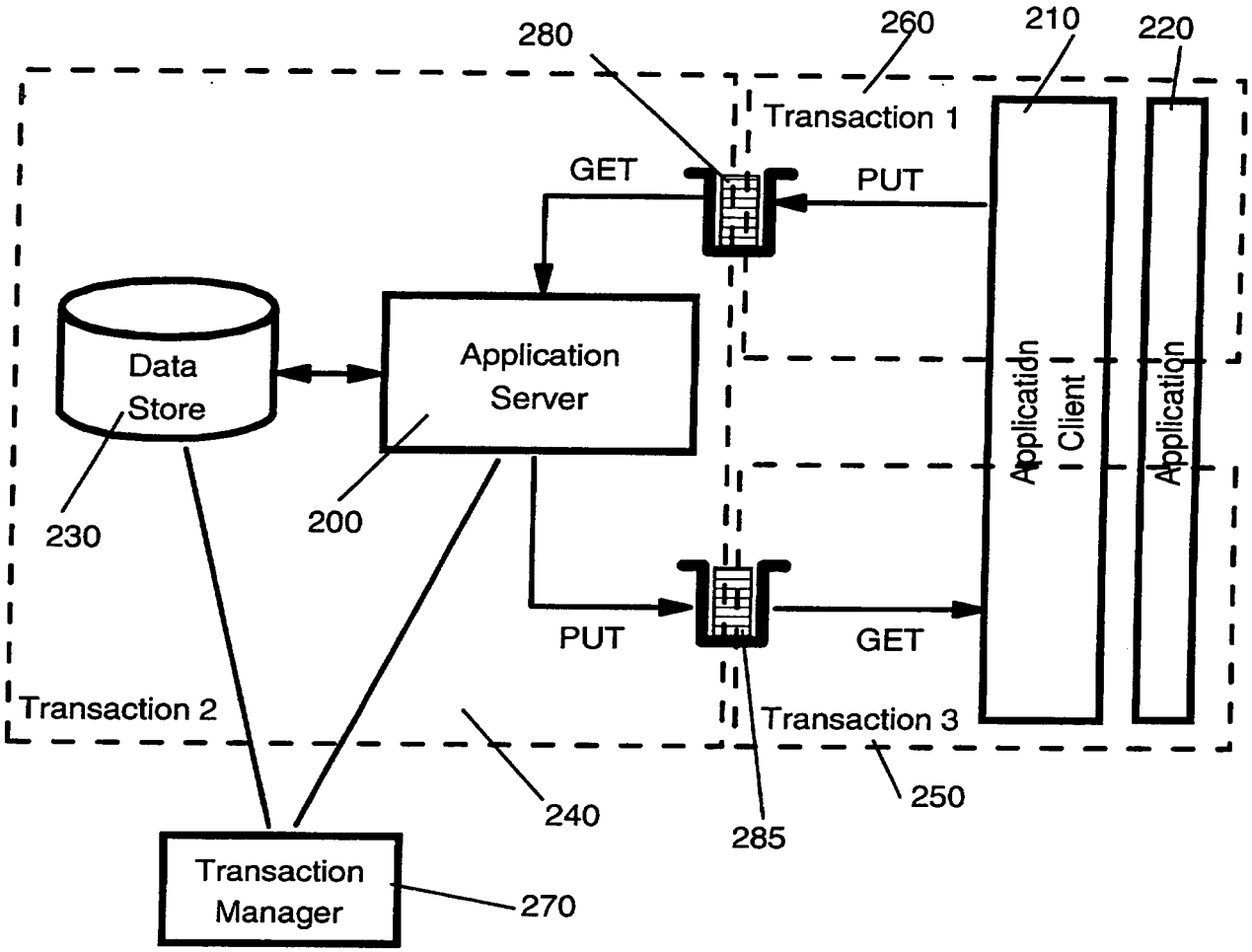
280
260
210
220
Transaction 1
GET
PUT
Data Store
Application Server
Application Client
Application
230
200
PUT
GET
Transaction 2
Transaction 3
240
250
285
Transaction Manager
270

FIG. 2

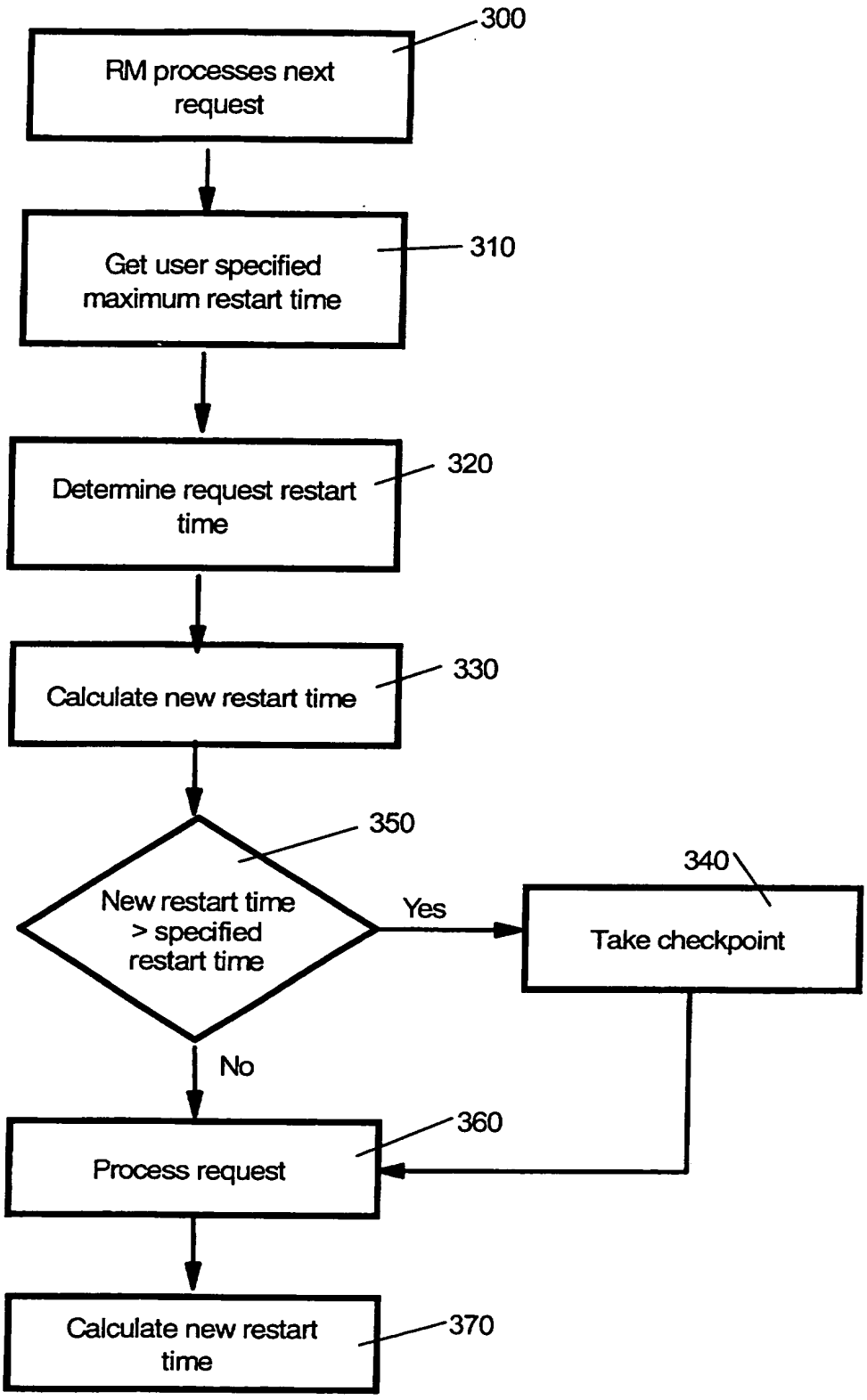

FIG. 3

NOV 2 1 2005.

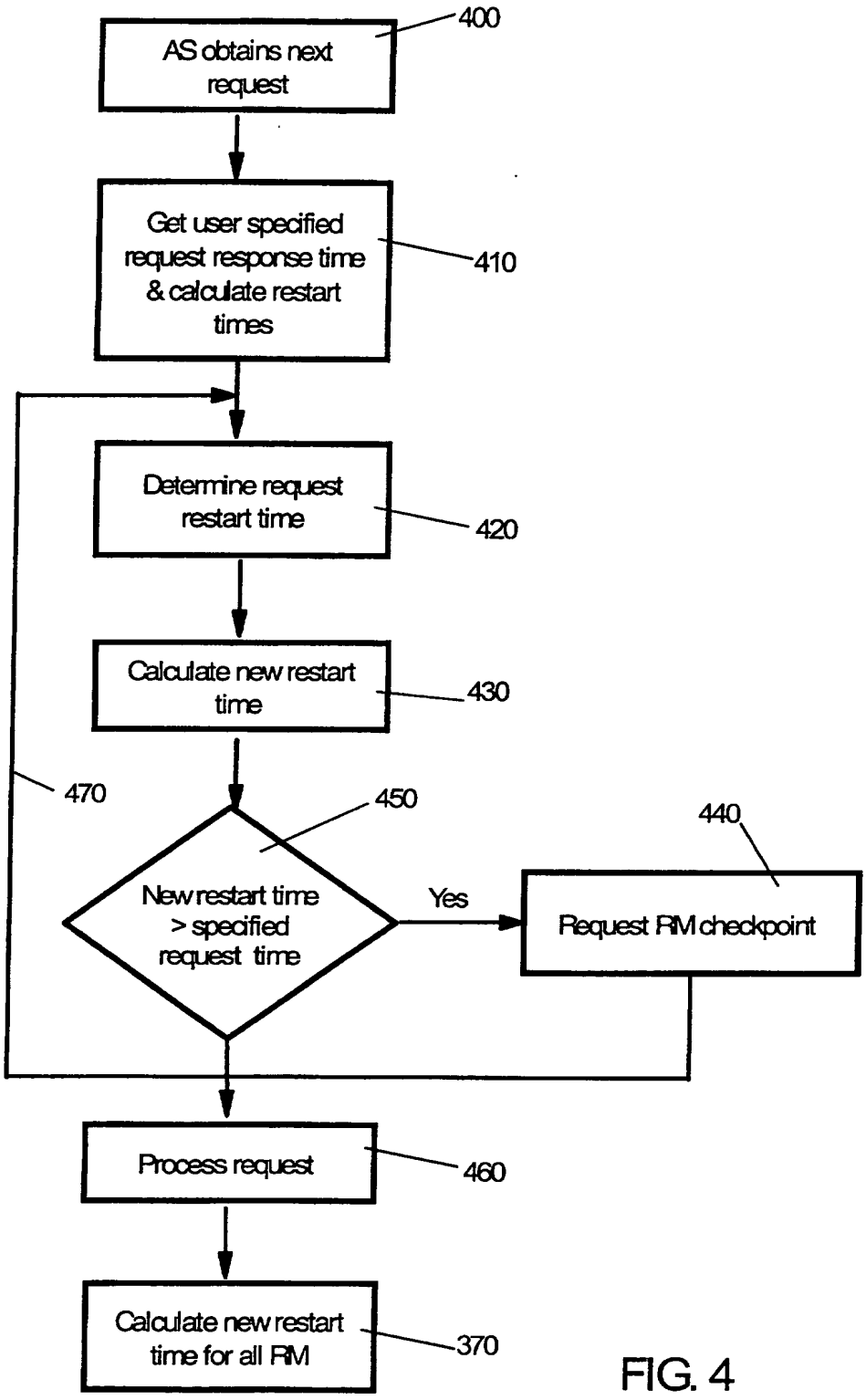
AS obtains next request
400
Get user specified request response time & calculate restart times
410
Determine request restart time
420
Calculate new restart time
430
470
450
New restart time > specified request time
Yes
440
Request RM checkpoint
Process request
460
Calculate new restart time for all RM
370

FIG. 4

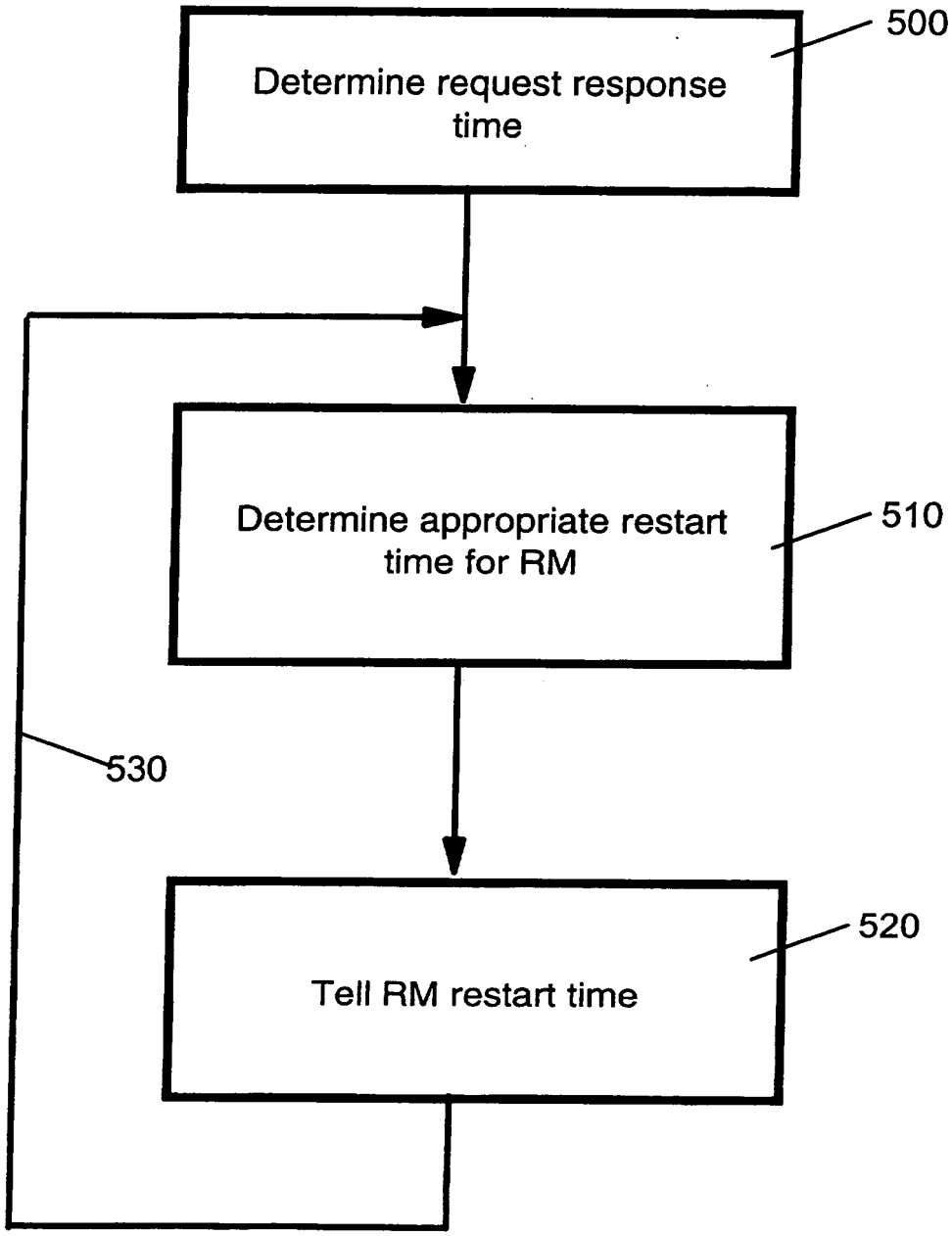

FIG. 5

NOV 2 1 2006

THIS PAGE BLANK (USPTO)